Reproducible research with R

26th June 2013

Outline

Bonus – figure building with `make`

- Introduction
  - The problem of dependencies
  - Automated figure building with `make`
- Basic use of `make`
  - Simple rules
  - Phony targets
- Going global
  - Using a `config.R` file
  - Wildcards, macros and patterns

Outline

Bonus – figure building with `make`

Outline

Bonus – figure building with `make`

- Introduction
  - The problem of dependencies
  - Automated figure building with `make`
- Basic use of `make`
  - Simple rules
  - Phony targets
- Going global
  - Using a `config.R` file
  - Wildcards, macros and patterns

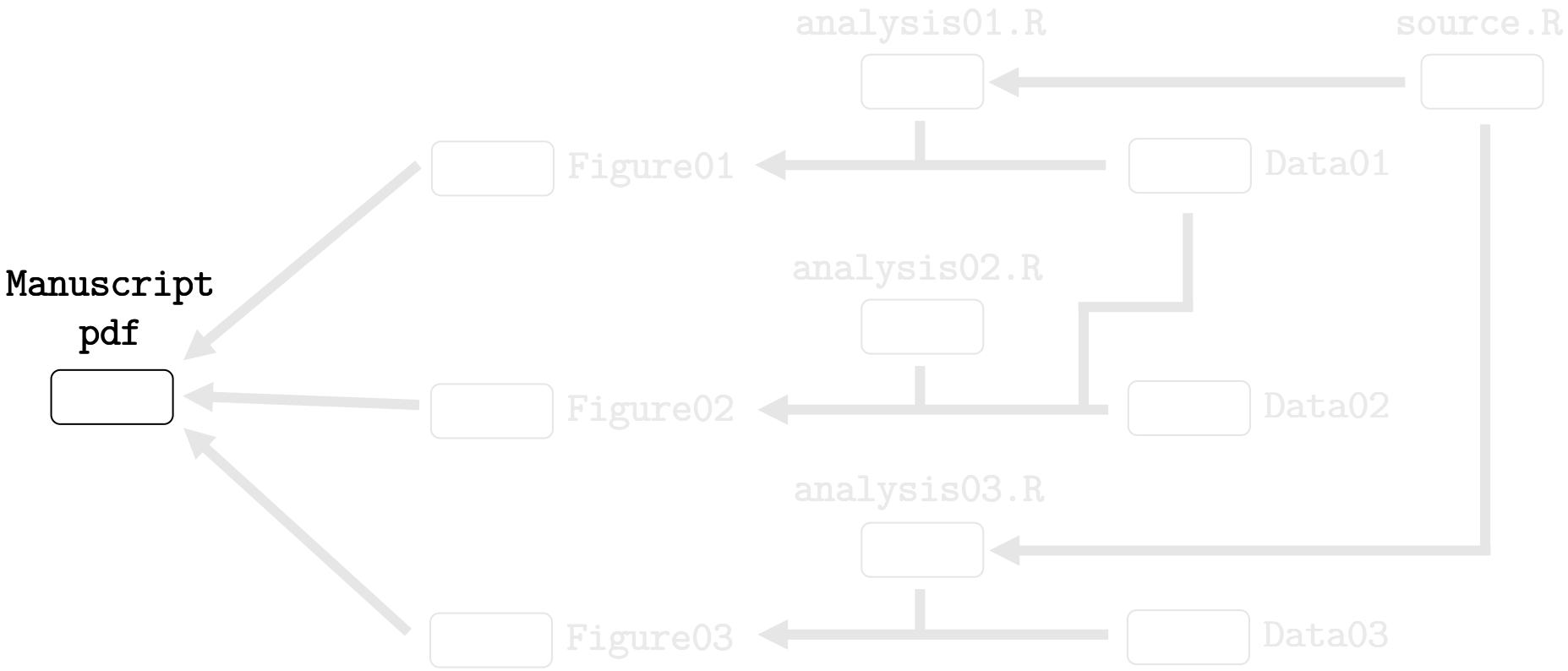
Outline

Bonus – figure building with `make`

- Introduction
  - The problem of dependencies
  - Automated figure building with `make`
- Basic use of `make`
  - Simple rules
  - Phony targets
- Going global
  - Using a `config.R` file
  - Wildcards, macros and patterns

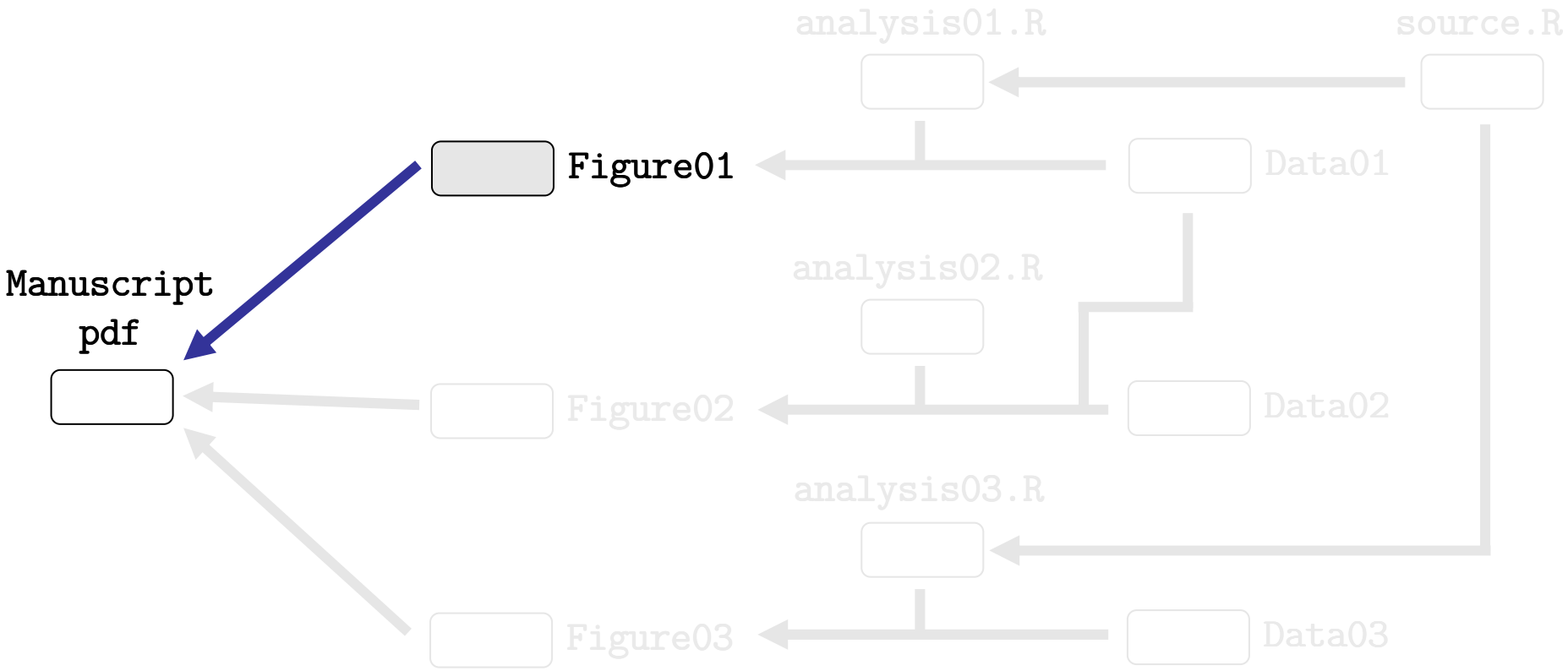
Bonus
Figure building with `make`

The problem of dependencies



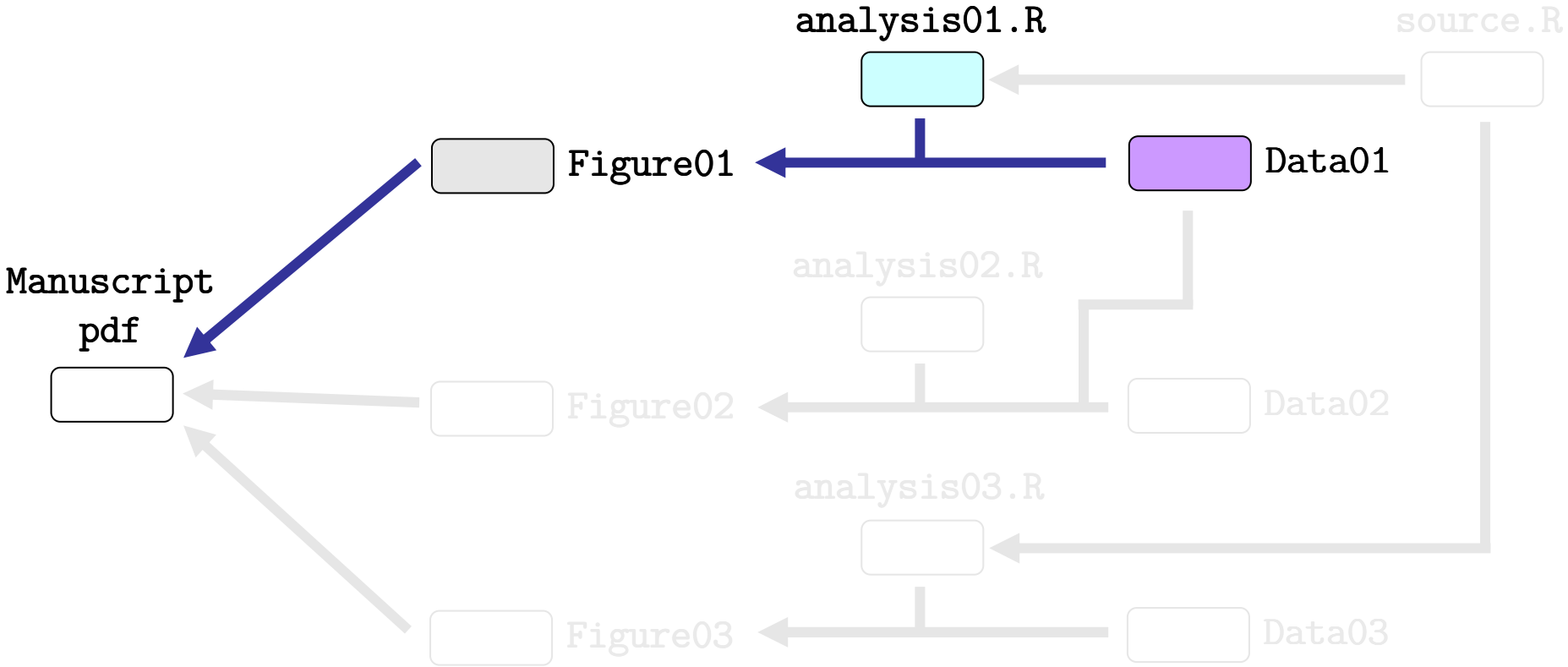
(shamelessly modified from http://software-carpentry.org/4_0/make/basics.html)

The problem of dependencies



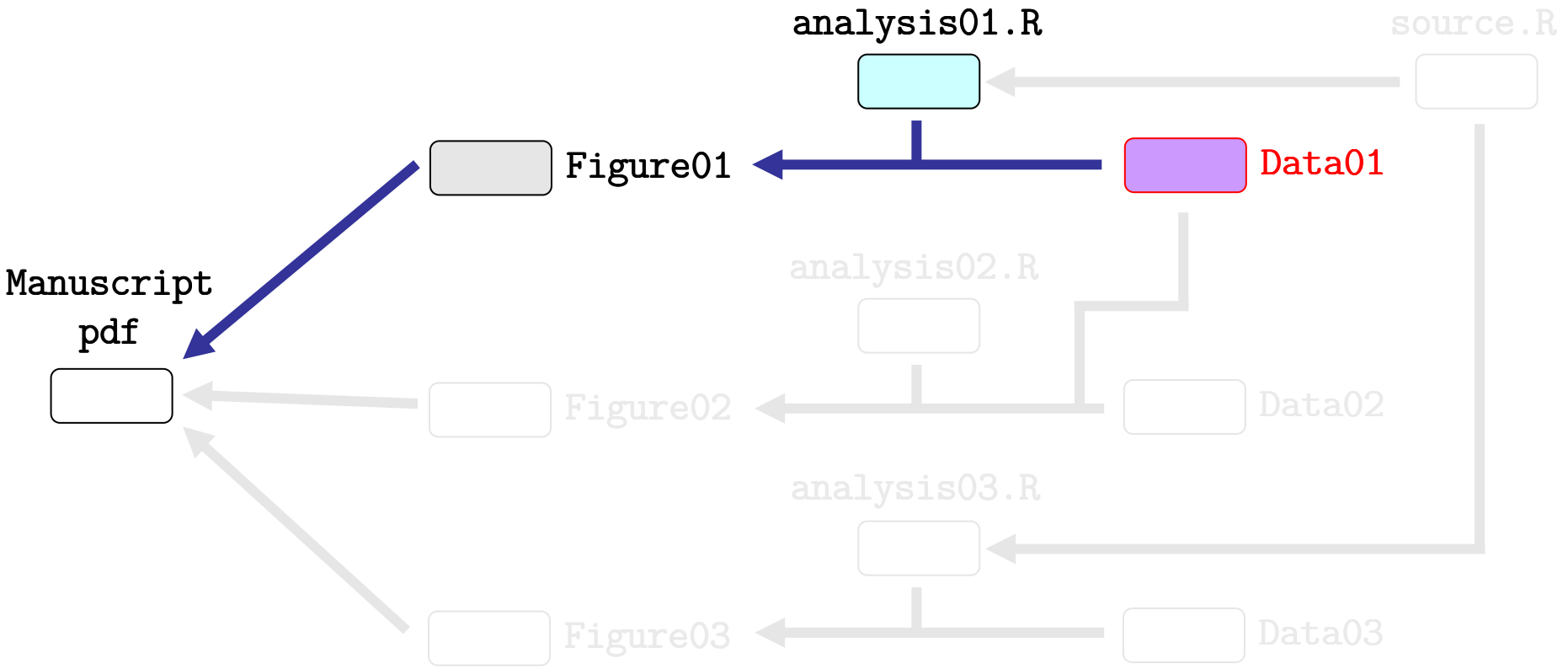
(shamelessly modified from http://software-carpentry.org/4_0/make/basics.html)

The problem of dependencies



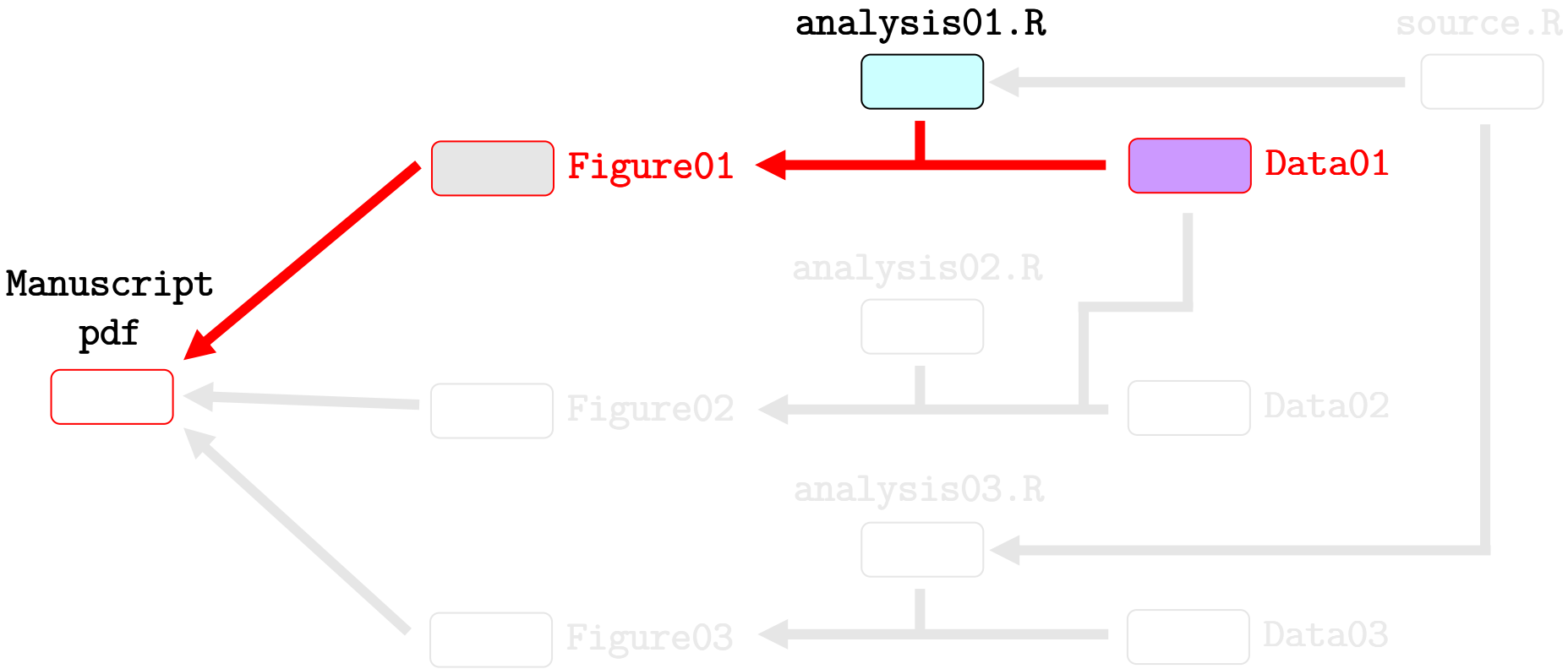
(shamelessly modified from http://software-carpentry.org/4_0/make/basics.html)

The problem of dependencies



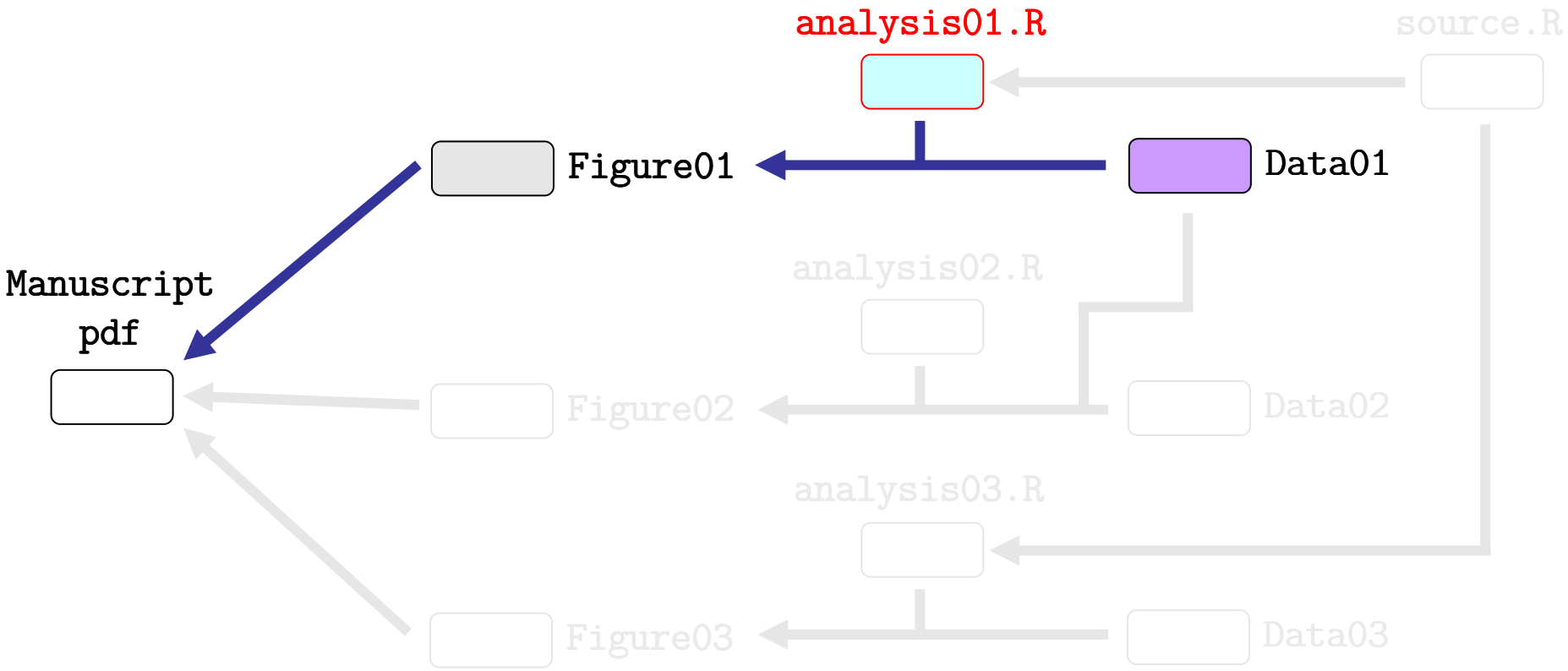
(shamelessly modified from http://software-carpentry.org/4_0/make/basics.html)

The problem of dependencies



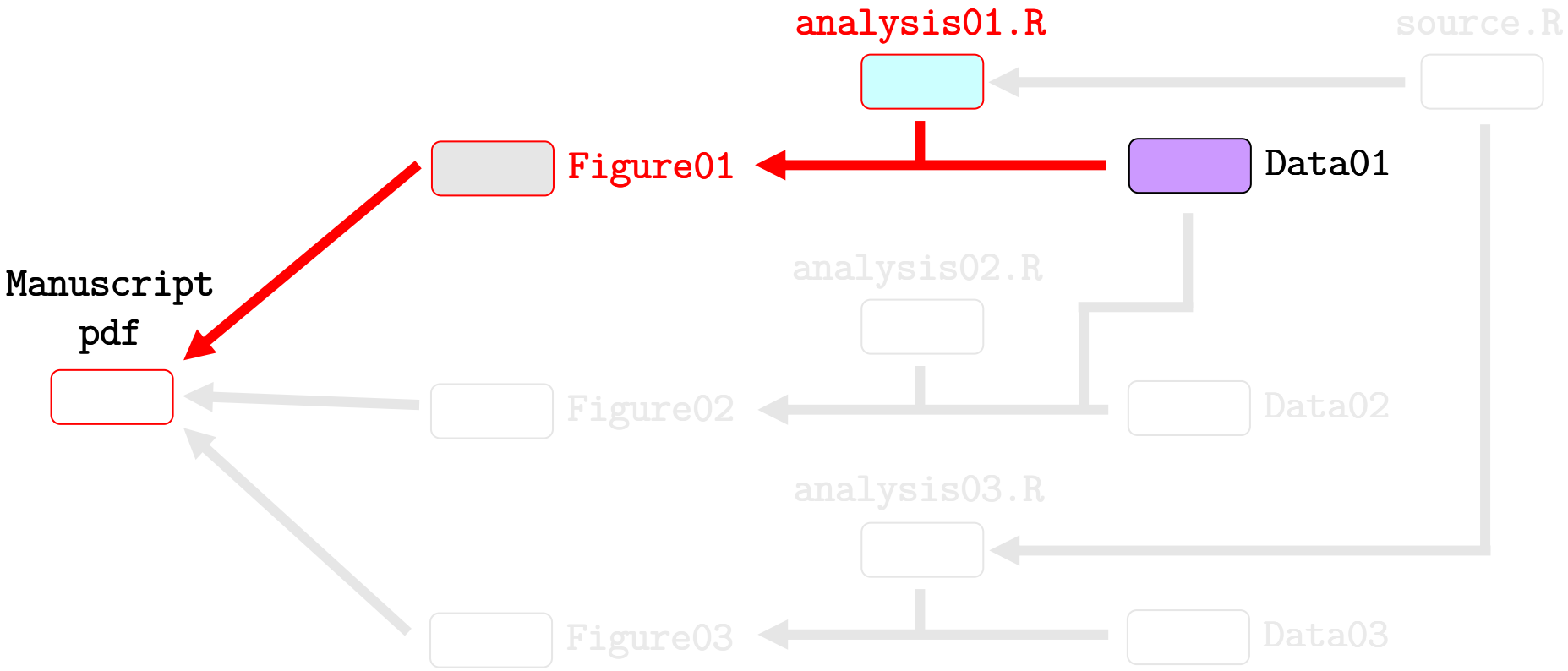
(shamelessly modified from http://software-carpentry.org/4_0/make/basics.html)

The problem of dependencies



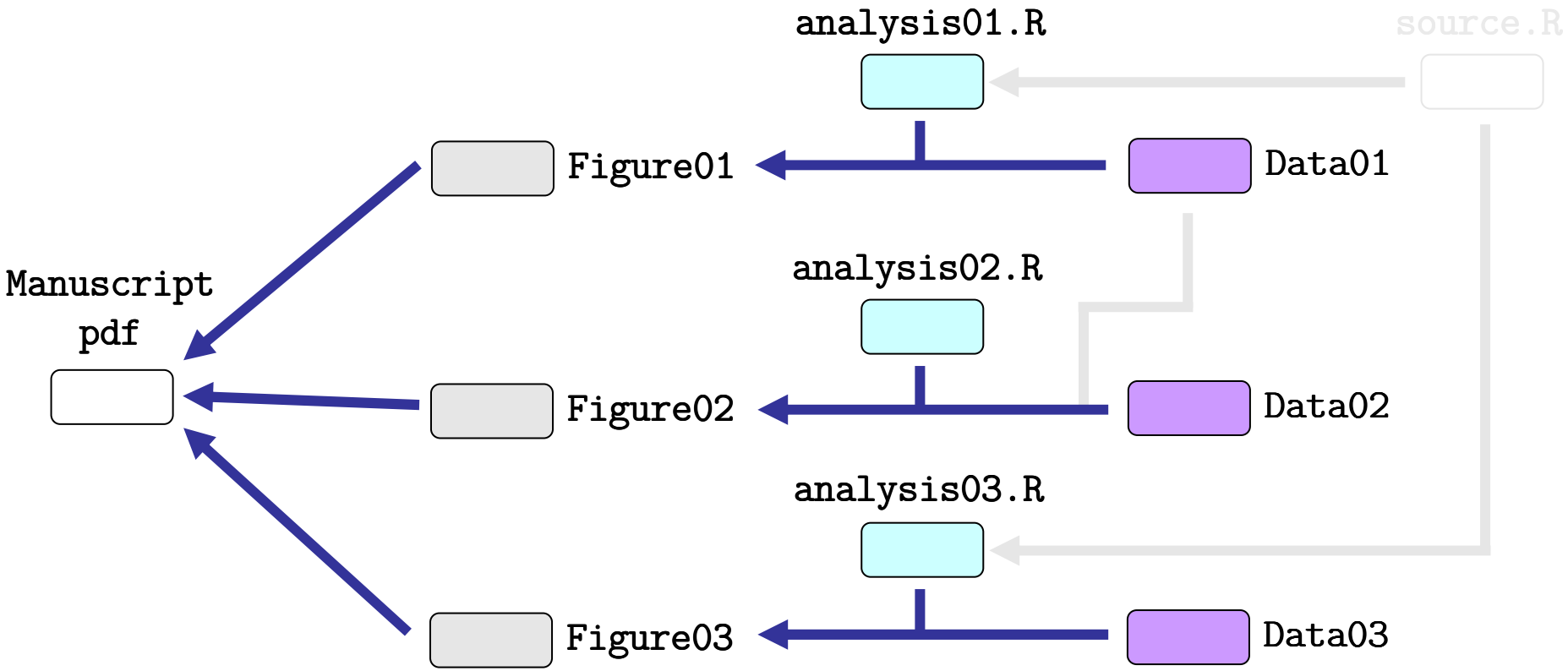
(shamelessly modified from http://software-carpentry.org/4_0/make/basics.html)

The problem of dependencies



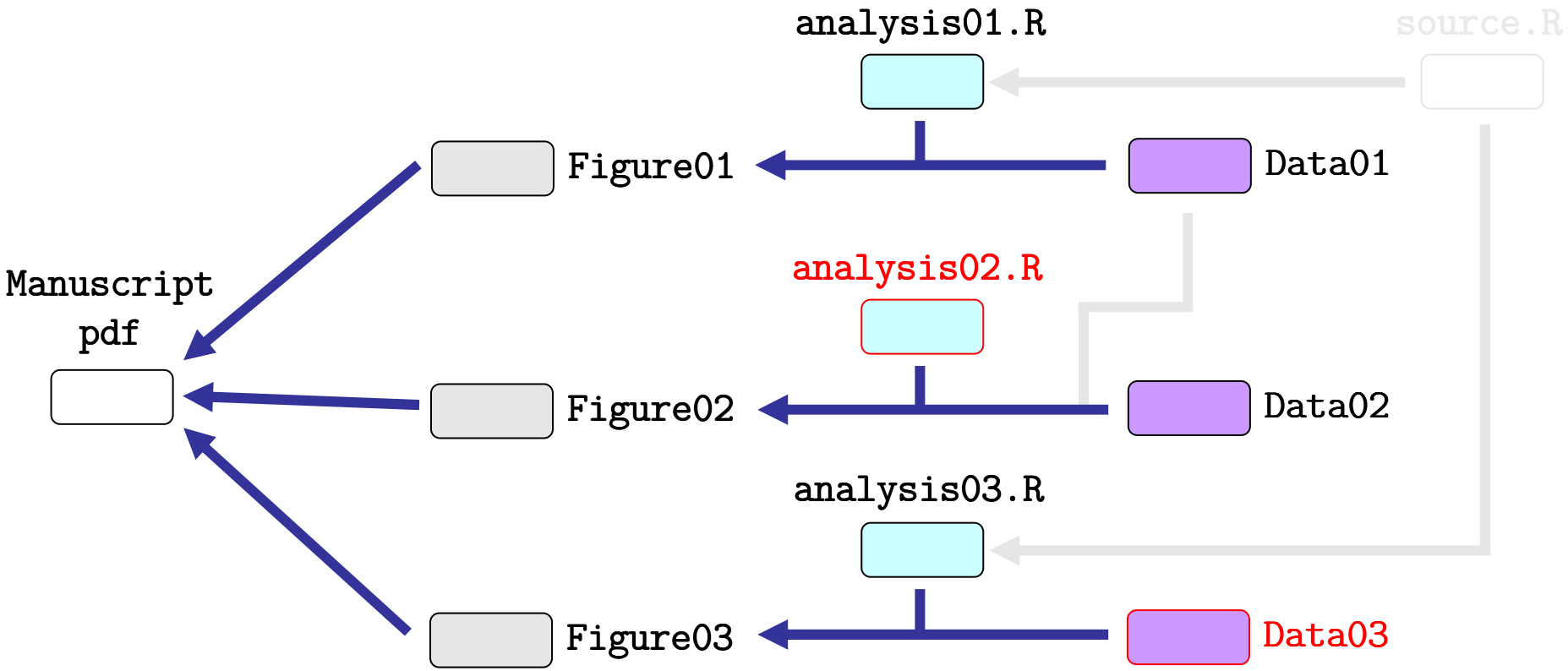
(shamelessly modified from http://software-carpentry.org/4_0/make/basics.html)

The problem of dependencies



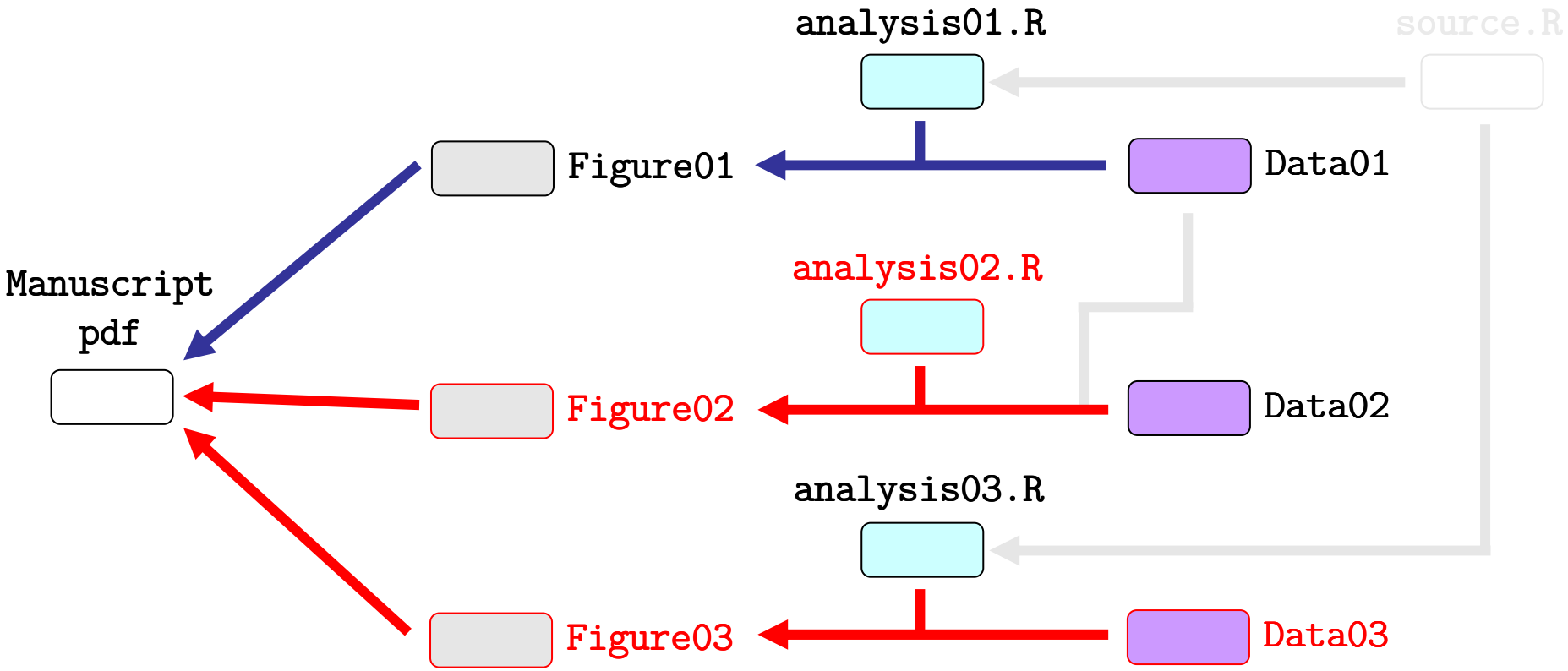
(shamelessly modified from http://software-carpentry.org/4_0/make/basics.html)

The problem of dependencies



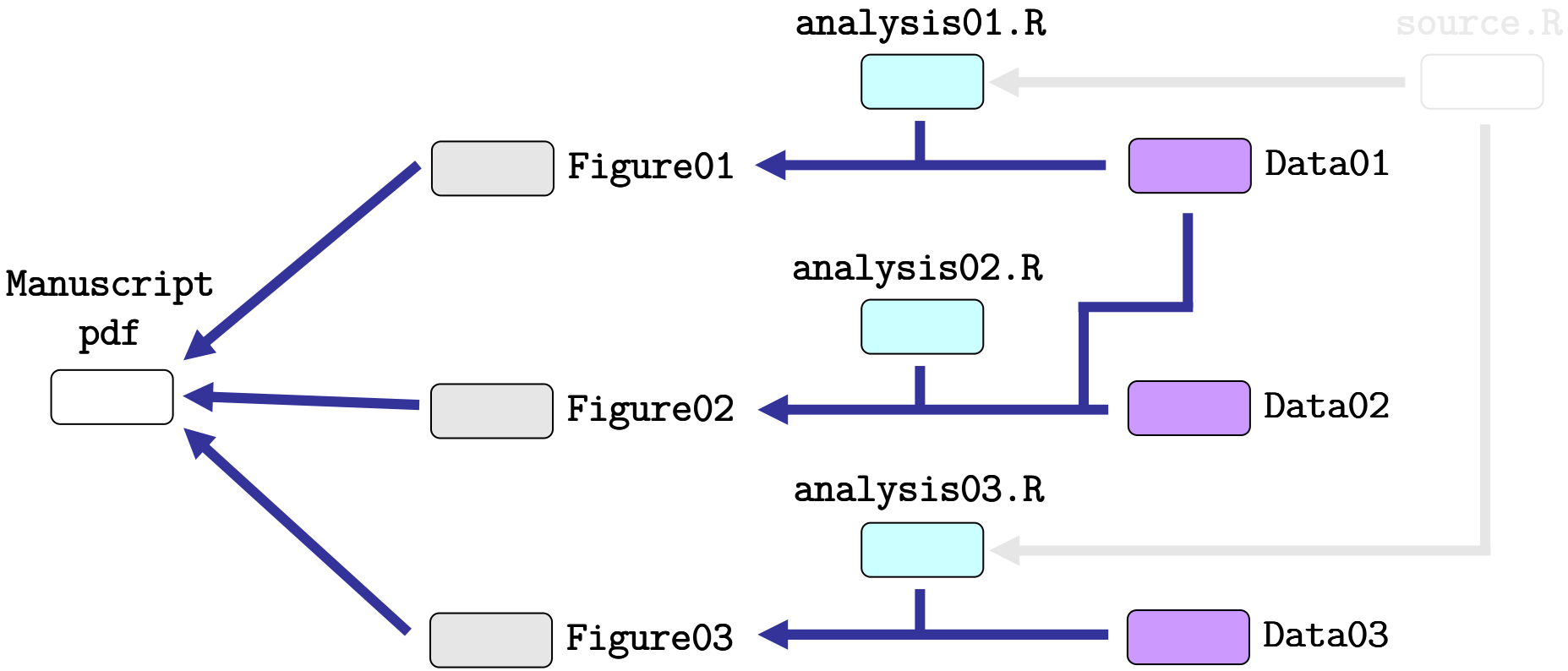
(shamelessly modified from http://software-carpentry.org/4_0/make/basics.html)

The problem of dependencies



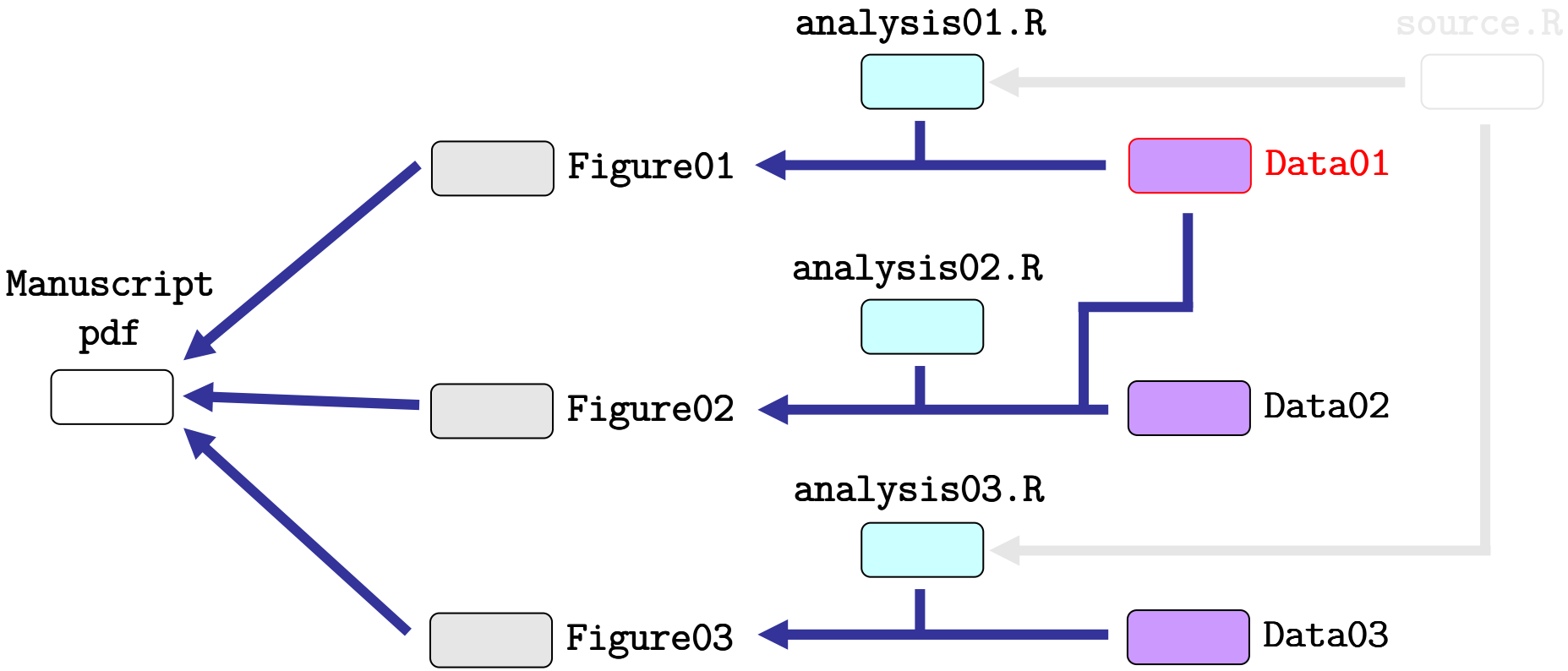
(shamelessly modified from http://software-carpentry.org/4_0/make/basics.html)

The problem of dependencies



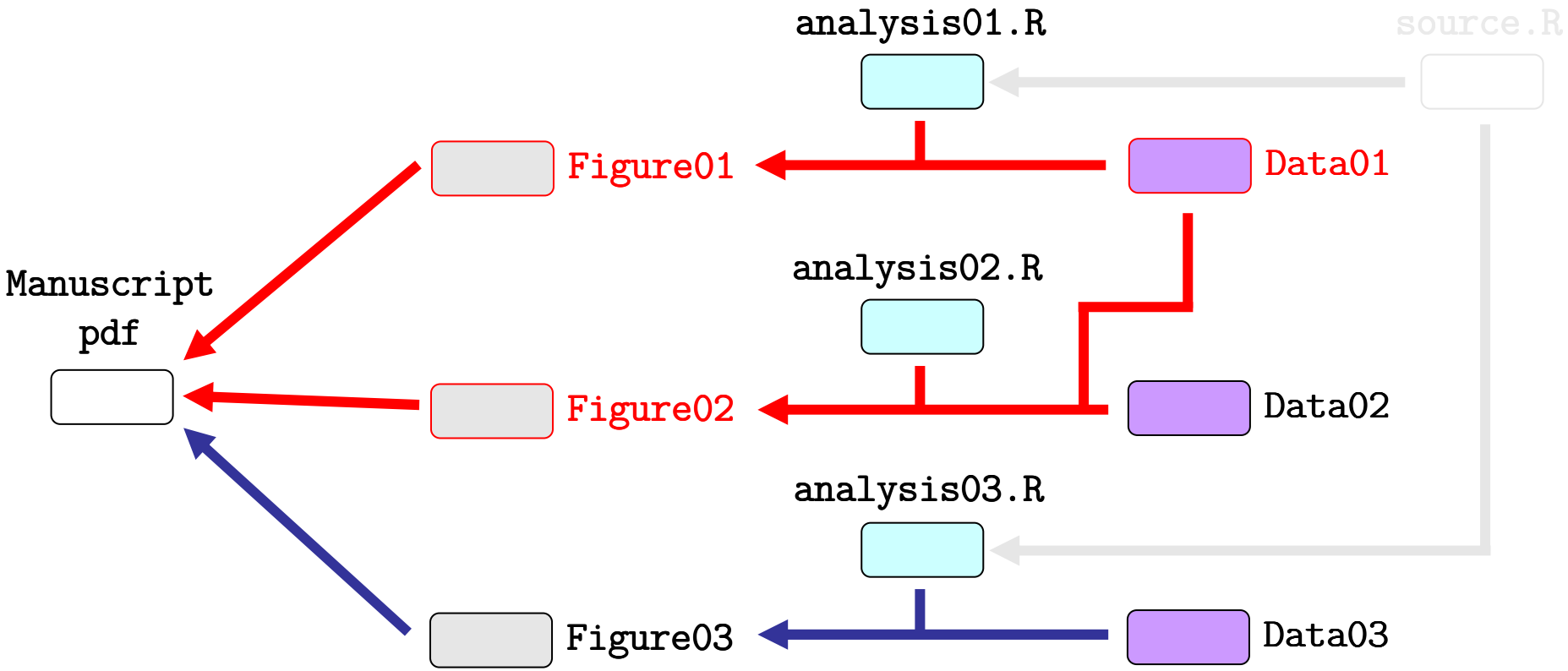
(shamelessly modified from http://software-carpentry.org/4_0/make/basics.html)

The problem of dependencies



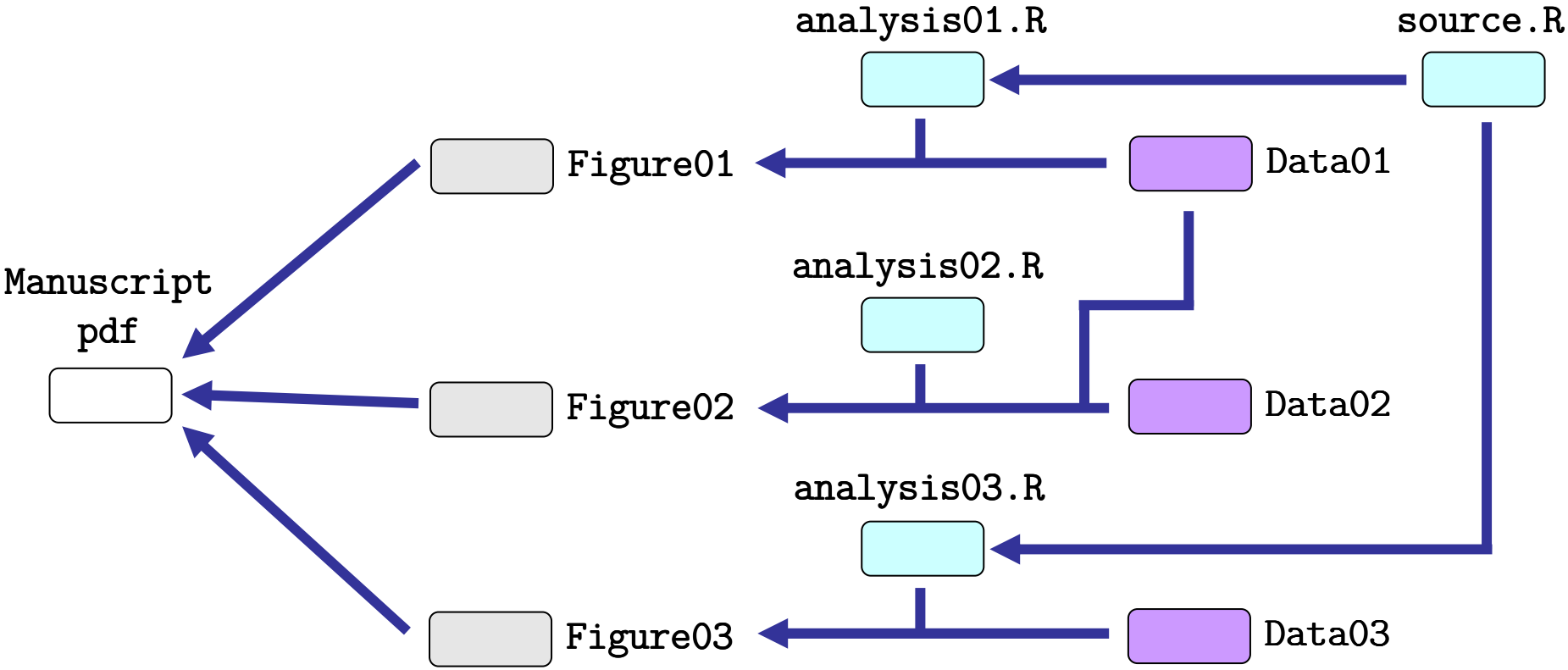
(shamelessly modified from http://software-carpentry.org/4_0/make/basics.html)

The problem of dependencies



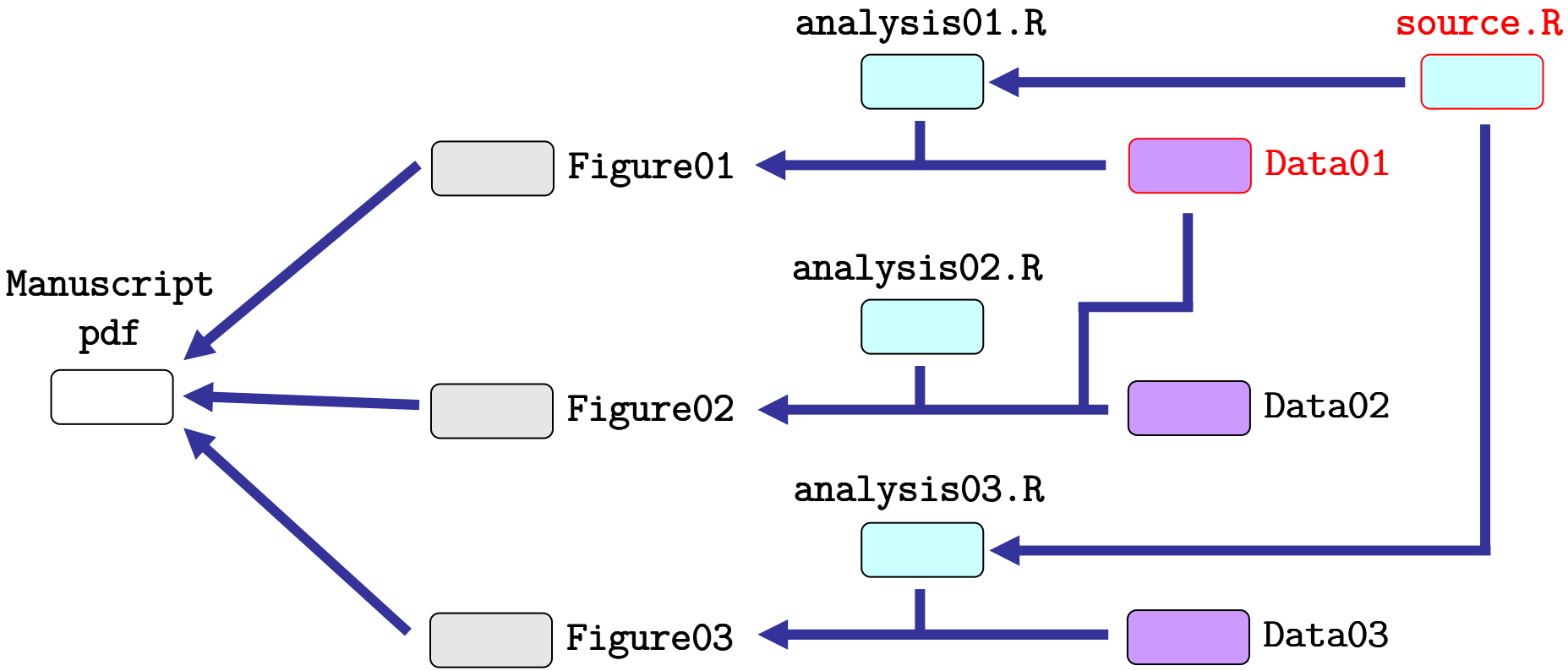
(shamelessly modified from http://software-carpentry.org/4_0/make/basics.html)

The problem of dependencies



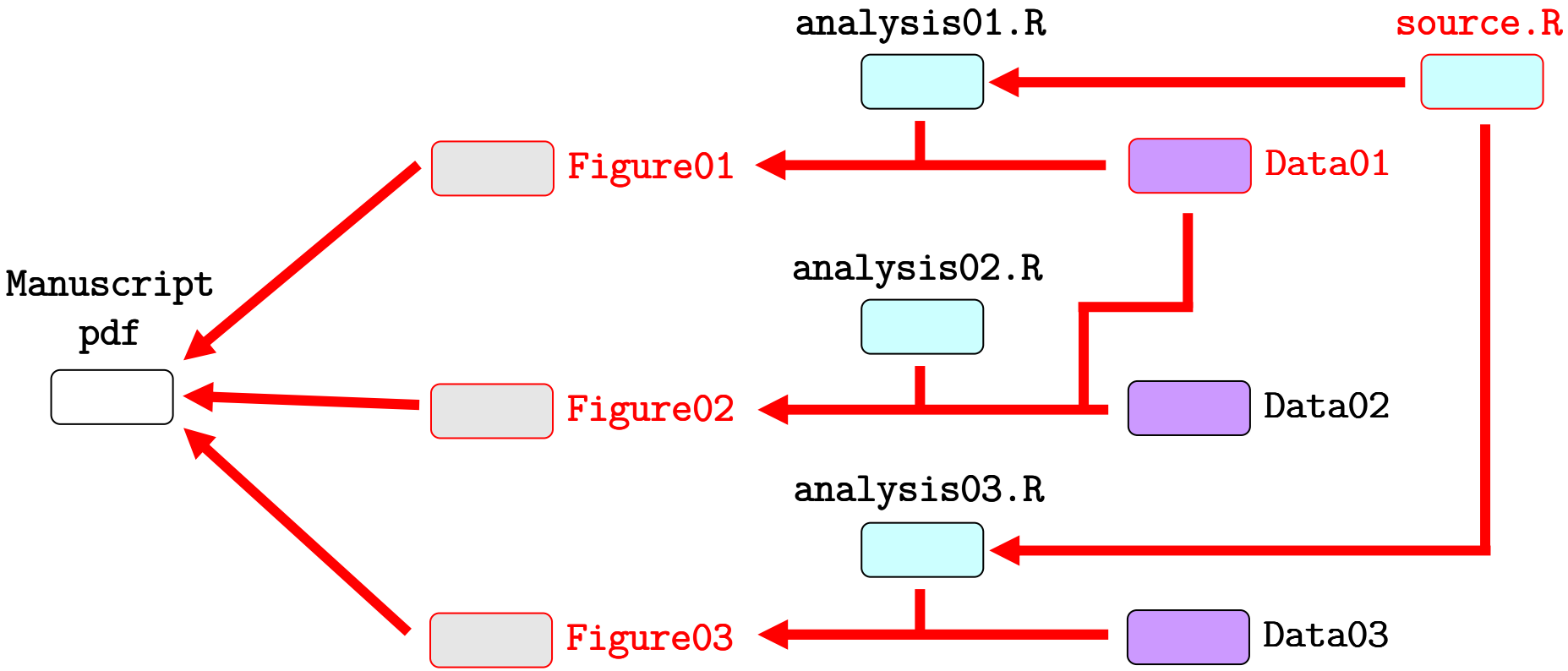
(shamelessly modified from http://software-carpentry.org/4_0/make/basics.html)

The problem of dependencies



(shamelessly modified from http://software-carpentry.org/4_0/make/basics.html)

The problem of dependencies



(shamelessly modified from http://software-carpentry.org/4_0/make/basics.html)

The problem of dependencies

- Manual update is time consuming.
- It is also error prone.
- "Anything worth repeating is worth automating." Greg Wilson, software-carpentry.org

The problem of dependencies

- Manual update is time consuming.
- It is also error prone.
- "Anything worth repeating is worth automating." Greg Wilson, software-carpentry.org

The problem of dependencies

- Manual update is time consuming.
- It is also error prone.
- "Anything worth repeating is worth automating." Greg Wilson, software-carpentry.org

Automated building: `make`

- There are tools to manage dependencies graphs and automatically build what needs to be built.
- `make` is one of them, but there are others.
- Automated building is simpler, faster and less error prone.

Automated building: `make`

- There are tools to manage dependencies graphs and automatically build what needs to be built.
- `make` is one of them, but there are others.
- Automated building is simpler, faster and less error prone.

Automated building: `make`

- There are tools to manage dependencies graphs and automatically build what needs to be built.
- `make` is one of them, but there are others.
- Automated building is simpler, faster and less error prone.

Automated building: `make`

- `make` is a very widely used tool.
- It is mature and free (`GNU Make`).
- Its syntax is somehow old-fashioned but it remains simple for basic use.

Automated building: `make`

- `make` is a very widely used tool.
- It is mature and free (`GNU Make`).
- Its syntax is somehow old-fashioned but it remains simple for basic use.

Automated building: `make`

- `make` is a very widely used tool.
- It is mature and free (`GNU Make`).
- Its syntax is somehow old-fashioned but it remains simple for basic use.

Basic use of `make`

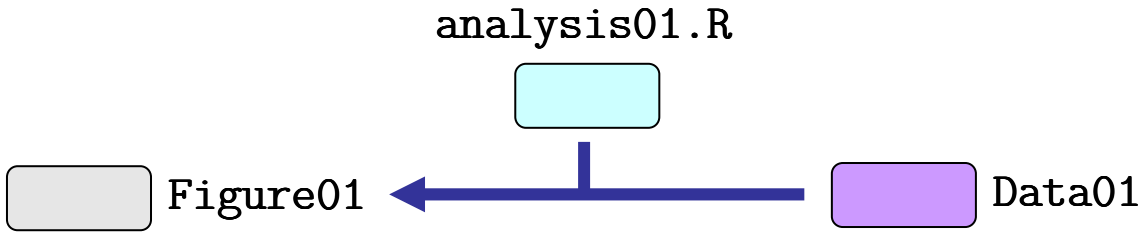
Practical

- `make` use will be introduced during a practical with RStudio.
- During the practical, we are going to generate four figures for a paper about Fiji earthquakes and Moomins.
- The first step is to have `make` running on our machine. This means adding the path to `make` to your `Path` environment variable if you are using Windows.

Modifying the PATH (Windows 7)

- Press the Windows key and the Pause key simultaneously.
- Click *'Advanced parameter settings > Environment variables > Path > Edit...'*.
- Add the path corresponding to your `make` installation, separated from the previous path by a semicolon.
- e.g.: C:\Program Files\GnuWin32\bin
- Click 'Ok'.

The makefile

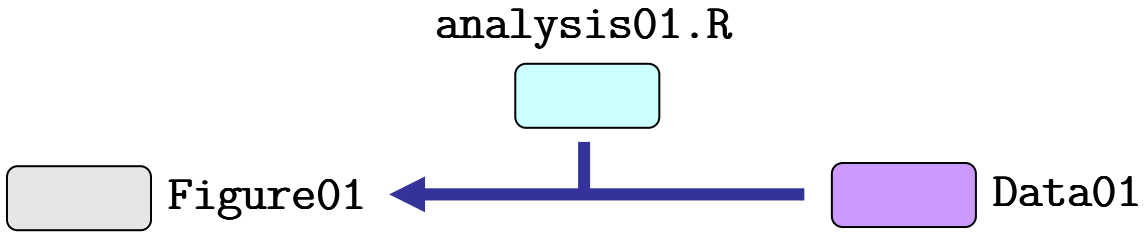



```
# makefile

Figure01 : Data01
    Rscript --vanilla analysis01.R Data01
```

- `make` knows about the dependencies between files by reading a **makefile**.
- A makefile describes the relations between files using **rules**.
- Each rule has 3 parts: a **target**, a **prerequisite** and an **action**.

The makefile

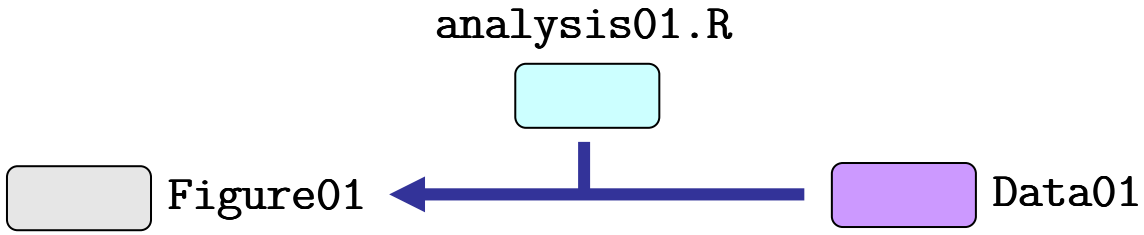



```
# makefile

Figure01 : Data01
    Rscript --vanilla analysis01.R Data01
```

- `make` knows about the dependencies between files by reading a **makefile**.
- A makefile describes the relations between files using **rules**.
- Each rule has 3 parts: a **target**, a **prerequisite** and an **action**.

The makefile

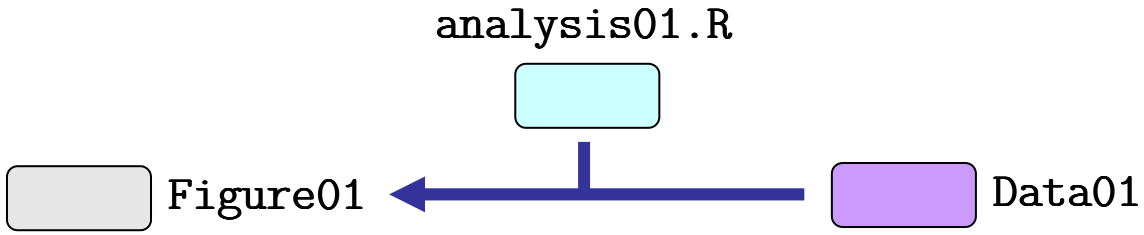



```
# makefile

Figure01 : Data01
    Rscript --vanilla analysis01.R Data01
```

- `make` knows about the dependencies between files by reading a **makefile**.
- A makefile describes the relations between files using **rules**.
- Each rule has 3 parts: a **target**, a **prerequisite** and an **action**.

The makefile

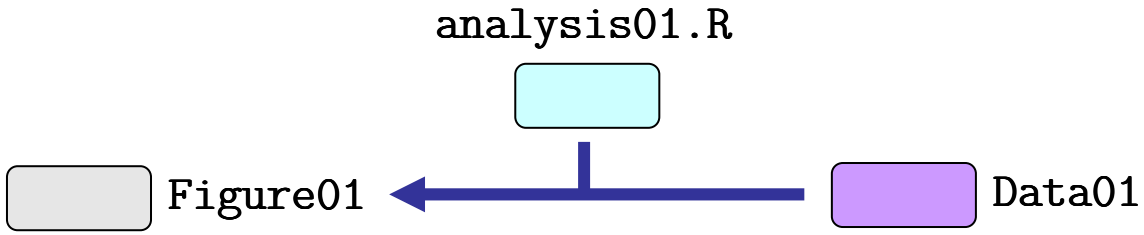



```
# makefile

Figure01 : Data01
    Rscript --vanilla analysis01.R Data01
```

- `make` knows about the dependencies between files by reading a **makefile**.
- A makefile describes the relations between files using **rules**.
- Each rule has 3 parts: a **target**, a **prerequisite** and an **action**.

The makefile

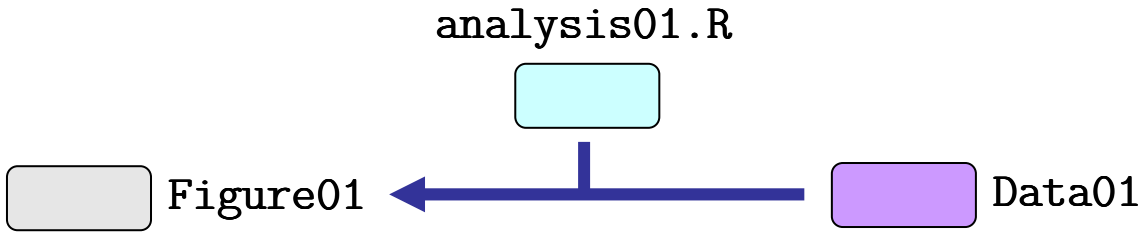



```
# makefile

Figure01 : Data01
    Rscript --vanilla analysis01.R Data01
```

- `make` knows about the dependencies between files by reading a **makefile**.
- A makefile describes the relations between files using **rules**.
- Each rule has 3 parts: a **target**, a **prerequisite** and an **action**.

The makefile

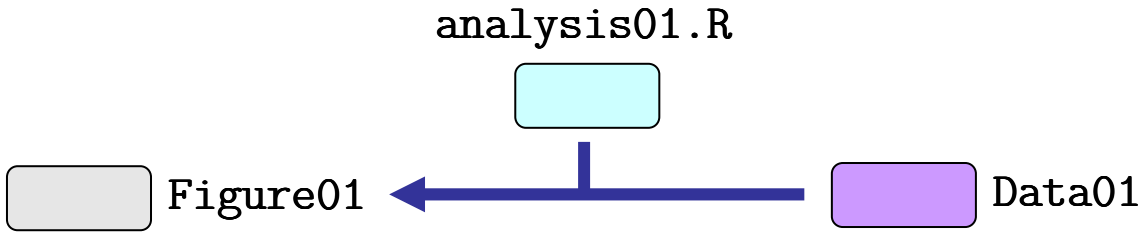



```
# makefile

Figure01 : Data01
    Rscript --vanilla analysis01.R Data01
```

- `make` knows about the dependencies between files by reading a **makefile**.
- A makefile describes the relations between files using **rules**.
- Each rule has 3 parts: a **target**, a **prerequisite** and an **action**.

The makefile



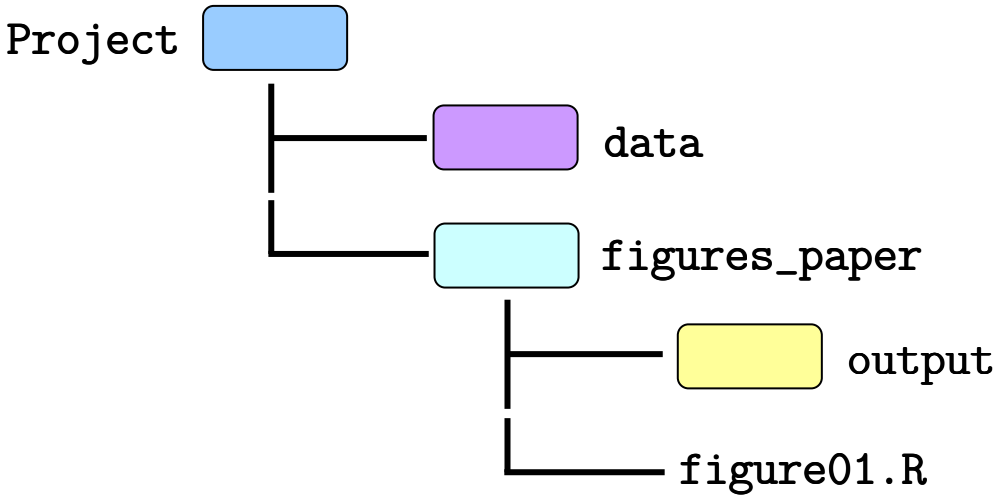

```
# makefile

Figure01 : Data01
    Rscript --vanilla analysis01.R Data01
```

- Comments starts with #.
- Actions have to start with a `tab` character.

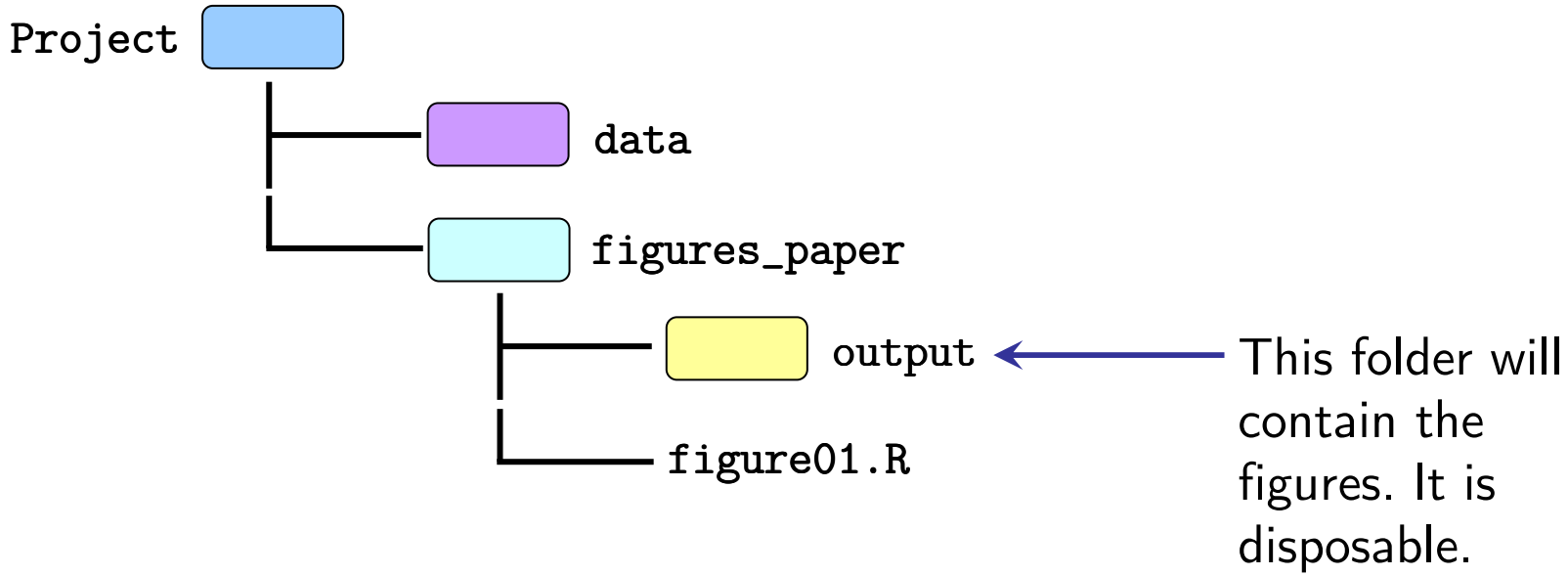
Ex.1: one file, one dependency

- Here is the structure of the folder used for the practical:



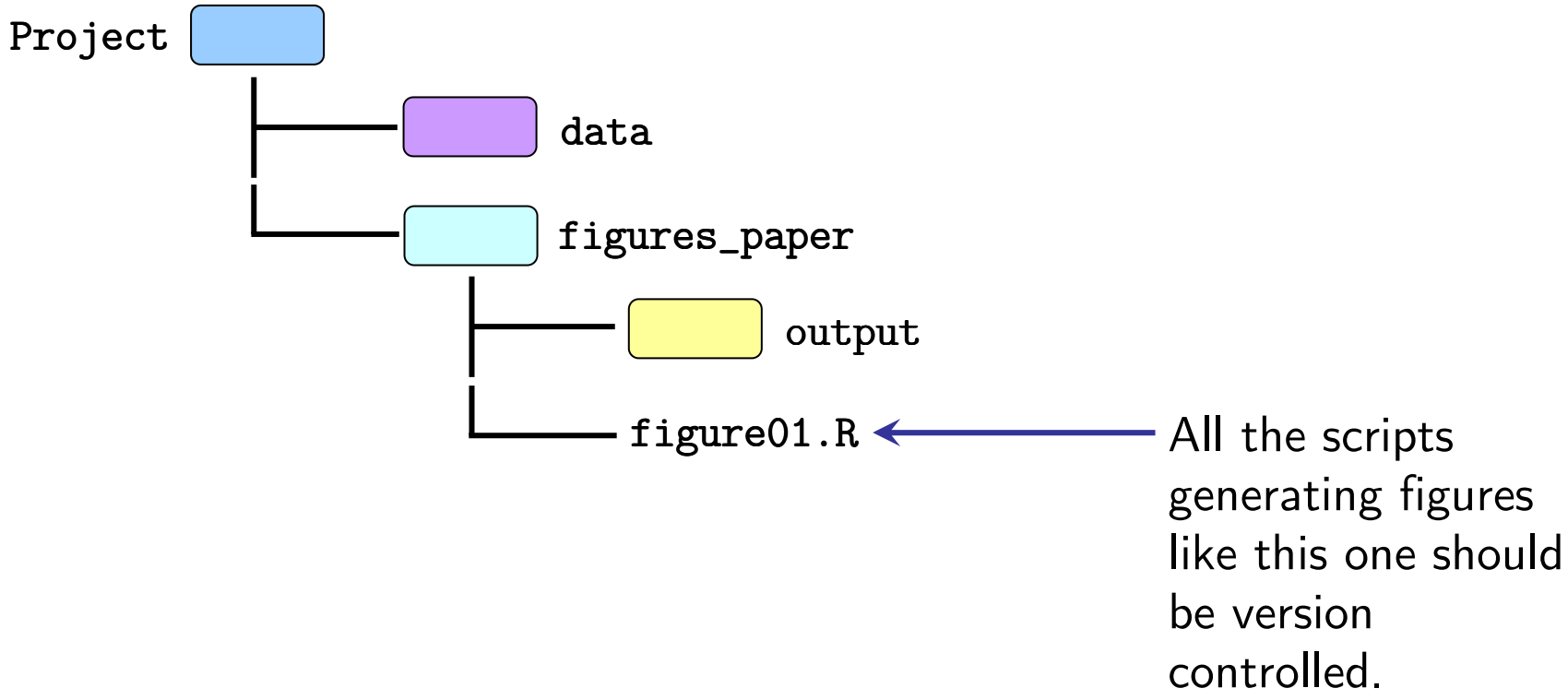
Ex.1: one file, one dependency

- Here is the structure of the folder used for the practical:



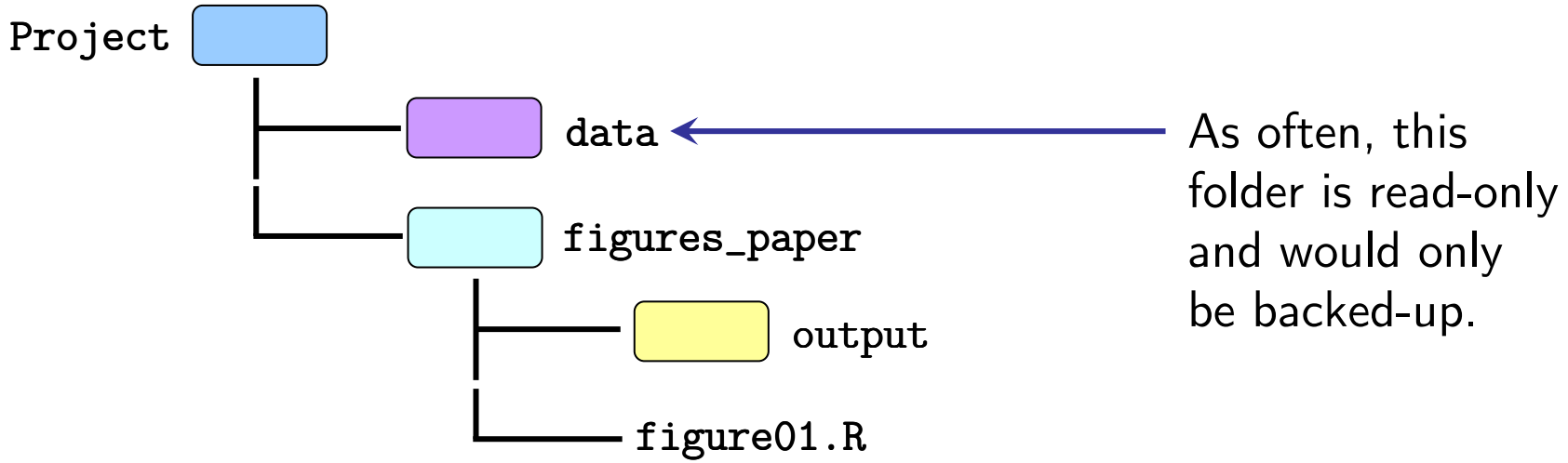
Ex.1: one file, one dependency

- Here is the structure of the folder used for the practical:



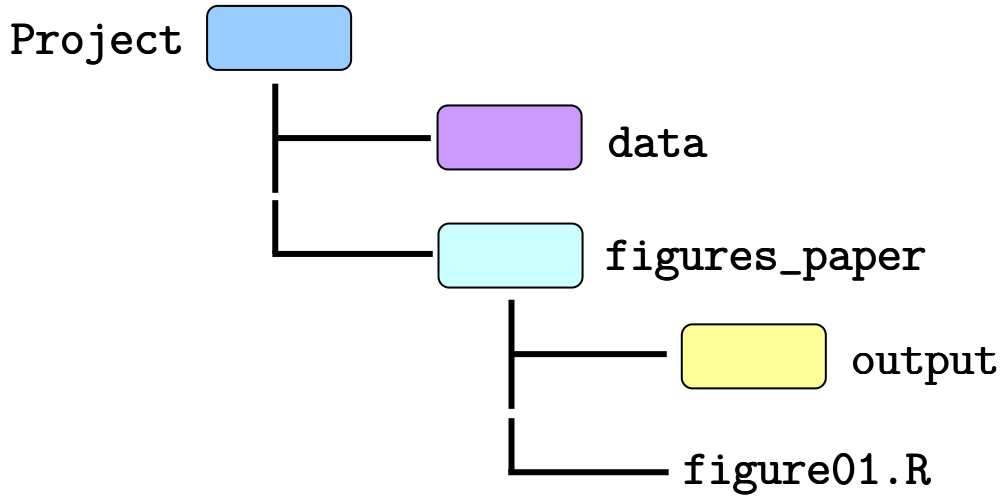
Ex.1: one file, one dependency

- Here is the structure of the folder used for the practical:



Ex.1: one file, one dependency

- Here is the structure of the folder used for the practical:



- As usual, reference files containing all the code we will use during the practical are available and stored in the `reference_files` folder.

Ex.1: one file, one dependency

- Let's start a new RStudio session and create an empty script saved as `figure01.R` in `./figures_paper`.
- Make sure your working directory is `./figures_paper`.
- Our first figure will use a dataset included in R. It does not depend on an external data file.

Ex.1: one file, one dependency

- We start with a very simple plot and run the code with `CTRL + SHIFT + S`:

```
# figure 01 - Map of earthquakes off Fiji
# using R dataset 'quakes'

### data ###

long = quakes$long
lat = quakes$lat
depth = quakes$depth

### plot ###

plot(long, lat, asp = 1) # asp = 1 determine the aspect ratio
```

Ex.1: one file, one dependency

- We add the `pdf` command to save the plot to a file.

```
# figure 01 - Map of earthquakes off Fiji
# using R dataset 'quakes'

### data ###

long = quakes$long
lat = quakes$lat
depth = quakes$depth

### plot ###

pdf("../output/figure01.pdf",
    width = 6, height = 4, pointsize = 12)

plot(long, lat, asp = 1) # asp = 1 determine the aspect ratio

dev.off()
```

Ex.1: one file, one dependency

- We add a flag to choose whether to display the figure or to send it to the pdf file.

```
### parameters ###

write.pdf = FALSE

### data ###
[...]

### plot ###

if (write.pdf) {
pdf("../output/figure01.pdf",
    width = 6, height = 4, pointsize = 12)
}

plot(long, lat, asp = 1) # asp = 1 determine the aspect ratio

if (write.pdf) {
  dev.off()
}
```

Ex.1: one file, one dependency

- We want to tune a bit the look of the plot. For this, it is nice to have all the relevant parameters at the beginning of the file:

```
### parameters ###

# --- file output ---
write.pdf = T

# --- limits and margins ---
margins = c(2, 3, 2, 0)
xlim = c(165, 190)
ylim = c(-40, -10)

# --- title and axes ---
title.label = "Locations of earthquakes off Fiji"
x.values.at = seq(165, 190, by = 5)
y.values.at = seq(-40, -10, by = 10)
xlab = "Longitude"
ylab = "Latitude"
```

Ex.1: one file, one dependency

- We then update the plot section:

```
### plot ###
#--- file output ---
if (write.pdf) {
pdf(file = "./output/figure01.pdf",
    width = 6, height = 4, pointsize = 12)
}
#--- empty plot ---
par(mar = margins) # set the outer margins
plot(0, 0, type = "n", xlim = xlim, ylim = ylim,
     xlab = "", ylab = "", axes = F, bty = "n",
     asp = 1) # asp = 1 determine the aspect ratio
#--- draw the points ---
points(long, lat, col = "blue", pch = 16, cex = 0.5)
#--- axes and title ---
axis(1, at = x.values.at)
axis(2, at = y.values.at, las = 1)
title(title.label)
#--- close the file ---
if (write.pdf) {
dev.off()
}
```

Ex.1: one file, one dependency

- We have a basic script generating a figure. Let's set `write.pdf` to `TRUE` and save the script.
- Create a file `figures.mk` in the `figures_paper` folder.
- We can write in it the rule to build `figure01.pdf`:

```
# makefile for figures for the paper
# "Fiji Earthquake and Moomin paintball"

output/figure01.pdf : figure01.R
        Rscript --vanilla figure01.R
```

- Note that the paths are relative to the location of the makefile.

Ex.1: one file, one dependency

- Open a terminal and go to the `figures_paper` folder.
- As this point, you might also have to add the path to `Rscript` to your `Path` environment variable.
- Run `make` and specify the makefile by typing:

```
make -f figures.mk
```

Ex.1: one file, one dependency

- Run `make` again. Nothing happens since the figure is up-to-date.

- Delete `figure01.pdf` and rerun `make`. The figure is automatically built because the file `figure01.pdf` does not exist yet.

Ex.1: one file, one dependency

- Let's add the axis label commands in our script:

```
### parameters ###
margins = c(3, 4, 2, 0)

### plot ###
#--- axes and title ---
mtext(xlab, 1, line = 2)
mtext(ylab, 2, line = 3)
```

- Let's run make again. The figure is updated because it depends on `figure01.R` and the dependency file is more recent than the target file `figure01.pdf`.

Ex.1: one file, one dependency

- `make` reads the makefile specified after -f:

```
make -f figures.mk
```

- If no file is specified, `make` will try to find a file called `makefile` or `Makefile`.

- This means we can be lazy by renaming `figures.mk` to `makefile`. We can then update the figure by typing only:

```
make
```

Ex.1: one file, one dependency

- Don't forget to set `write.pdf` to `TRUE` in the R script before running `make`, otherwise the figure file will never be updated.
- We will see later how to make things a bit simpler.

Ex.2: two files, one dependency each

- The previous case was easy. Using `make` does not really improve things a lot.
- We can make things more interesting by adding another figure, which also only depends on only one file.
- Copy the file `figure02.R` from the `reference_files` folder into the `figures_paper` folder.

Ex.2: two files, one dependency each

- The previous case was easy. Using `make` does not really improve things a lot.
- We can make things more interesting by adding another figure, which also only depends on only one file.
- Copy the file `figure02.R` from the `reference_files` folder into the `figures_paper` folder.

Ex.2: two files, one dependency each

- The previous case was easy. Using `make` does not really improve things a lot.
- We can make things more interesting by adding another figure, which also only depends on only one file.
- Copy the file `figure02.R` from the `reference_files` folder into the `figures_paper` folder and open it with RStudio.

Ex.2: two files, one dependency each

- You might need to install the RgoogleMaps package into RStudio.
- Run the code within RStudio. If everything works fine, set `write.pdf` to `TRUE`.
- Now we are going to update our makefile.

Ex.2: two files, one dependency each


```
# makefile for figures for the paper
# "Fiji Earthquake and Moomin paintball"

output/figure01.pdf : figure01.R
        Rscript --vanilla figure01.R

output/figure02.pdf : figure02.R
        Rscript --vanilla figure02.R
```

- Delete the content of the `output` folder and run `make`.
- Only `figure01.pdf` is produced. Why?
- By default, `make` only applies the first rule it finds in the makefile.

Ex.2: two files, one dependency each

- We can tell make to look for a specific rule:

```
make target
make output/figure02.pdf
```

- But this doesn't solve the problem: how can we tell `make` that it has to update several targets?

Ex.2: two files, one dependency each

- The solution is to use a **phony target**. A phony target is a target which is not a file: it is never up-to-date and will always result in an update of its dependencies.

```
# makefile for figures for the paper
# "Fiji Earthquake and Moomin paintball"

all : output/figure01.pdf output/figure02.pdf

output/figure01.pdf : figure01.R
        Rscript --vanilla figure01.R

output/figure02.pdf : figure02.R
        Rscript --vanilla figure02.R
```

Ex.2: two files, one dependency each

- We can run make by typing:

```
make all # or
make     # if all is the first target in the file
```

- `all` is never up-to-date since there is no such file.
- `make` decides to update it: for this it must first apply the rules concerning the prerequisites for `all` (if any) and then execute the action specified for `all`.
- `make` applies the rules for `figure01.pdf` and `figure02.pdf`. Since there is no action specified for `all`, it stops here.

Ex.3: more than one dependency

- We will build a figure which depends on more than one file.
- Copy the file `figure03.R` from the `reference_files` folder to the `figures_paper` folder and open it with RStudio.

Ex.3: more than one dependency

- Update the makefile:

```
# makefile for figures for the paper
# "Fiji Earthquake and Moomin paintball"

all : output/figure01.pdf output/figure02.pdf output/figure03.pdf

output/figure01.pdf : figure01.R
        Rscript --vanilla figure01.R

output/figure02.pdf : figure02.R
        Rscript --vanilla figure02.R

output/figure03.pdf : figure03.R ../data/paintball_data
        Rscript --vanilla figure03.R
```

Ex.3: more than one dependency

- Run `make`.
- Delete the contents of `output` and run `make` again.
- You can play around by modifying slightly the scripts and the data file and running `make`.

Another phony target: clean

- `make` can be used for other things than just building files.
- It can be used to clean a directory structure before running `make all` for example.
- `clean` is a common phony target. Let's add it at the end of the makefile (here is a Windows version of the action to remove the output files):

```
clean :
        rmdir /s /q output
        mkdir output
```

Another phony target: clean

- We can clean our directory tree and rerun `make` from the beginning by typing:

```
make clean
make
```

- Any action that can be specified from the command line can be done by `make`.

Going global

Going global

- The philosophy of using `make` is to manage the building process at a global level.
- This means we want to minimize the amount of tuning which is specific to each file.
- We can use a `config.R` file which will be sourced by each script to be able to change global settings in one place.

Going global

- The philosophy of using `make` is to manage the building process at a global level.
- This means we want to minimize the amount of tuning specific to each file.
- We can use a `config.R` file which will be sourced by each script to be able to change global settings in one place.

Going global

- The philosophy of using `make` is to manage the building process at a global level.
- This means we want to minimize the amount of tuning specific to each file.
- We can use a `config.R` file which will be sourced by each script to be able to change global settings in one place.

Refactoring the code

- Let's start by modifying our existing code. In `figure01.R`, add the code to map depth to a color gradient:

```
### parameters ###
#--- colors ---
depth.ncol = 100
depth.col.shallow = "lightblue"
depth.col.deep = "darkred"

### plot ###
#--- prepare the color palette for depth ---
color.gradient = colorRampPalette(c(depth.col.shallow, depth.col.deep))
depth.colors = color.gradient(depth.ncol)
#--- draw the points ---
points(long, lat, col = depth.colors[cut(depth, depth.ncol)], pch = 16,
cex = 0.5)
```

- Run the code after setting `write.pdf` to `FALSE`.

Refactoring the code

- We would like to be able to change the color of the depth gradient in figure 1 and 2 by specifying it in only one place.

- A good way to do it is to create a `config.R` file in the `figures_paper` folder:

```
### Global settings ###

#--- depth color gradient ---
cfg.depth.col.shallow = "lightblue"
cfg.depth.col.deep = "darkred"
```

Refactoring the code

- We source this file into figure01.R and figure02.R and update the color gradient variables in those:

```
# (at the beginning of each file)
source("./config.R")

### parameters ###
#--- colors ---
depth.ncol = 100
depth.col.shallow = cfg.depth.col.shallow
depth.col.deep = cfg.depth.col.deep
```

Refactoring the code

- We mustn't forget to update the makefile with the new dependencies:

```
# makefile for figures for the paper
# "Fiji Earthquake and Moomin paintball"

all : output/figure01.pdf output/figure02.pdf output/figure03.pdf

output/figure01.pdf : figure01.R config.R
        Rscript --vanilla figure01.R

output/figure02.pdf : figure02.R config.R
        Rscript --vanilla figure02.R

output/figure03.pdf : figure03.R ../data/paintball_data
        Rscript --vanilla figure03.R
```

Refactoring the code

- Now let's change the colors in the gradient (`config.R`) and see if make updates the figures:

```
### Global settings ###

#--- depth color gradient ---
cfg.depth.col.shallow = "pink"
cfg.depth.col.deep = "purple"
```

```
make clean
make
```

- The idea is to use this for all that is common to several figures: `cex`, `xlim`, `pdf.width`, `pdf.height`, etc...

Going global in the makefile

- The makefile itself can become cumbersome to manage when there is a lot of dependencies with similar names and patterns.
- It can be simplified by using wildcards and other mechanisms specific to `make`.
- Let's try to improve our makefile.

Going global in the makefile

- The makefile itself can become cumbersome to manage when there is a lot of dependencies with similar names and patterns.
- It can be simplified by using wildcards and other mechanisms specific to `make`.
- Let's try to improve our makefile.

Going global in the makefile

- The makefile itself can become cumbersome to manage when there is a lot of dependencies with similar names and patterns.
- It can be simplified by using wildcards and other mechanisms specific to `make`.
- Let's add a fourth figure showing Moomin's path after drinking a few glasses of Salmiakki.
- Copy the file `figure04.R` from the `reference_files` folder to the `figures_paper` folder.

Makefile: wildcards

- `figure04.R` takes as arguments data files recording Moomin's position on his island in Naantali.
- It can make as many records as we want.

```
Rscript --vanilla figure04.R ../data/moomin_walk_01.dat ../data/moomin_walk_02.dat
```

Makefile: wildcards

- Let's update the makefile:

```
# makefile for figures for the paper
# "Fiji Earthquake and Moomin paintball"

all : output/figure01.pdf output/figure02.pdf output/figure03.pdf output/figure04.pdf

output/figure01.pdf : figure01.R config.R
	Rscript --vanilla figure01.R

output/figure02.pdf : figure02.R config.R
	Rscript --vanilla figure02.R

output/figure03.pdf : figure03.R ../data/paintball_data
	Rscript --vanilla figure03.R

output/figure04.pdf : figure04.R ../data/moomin_walk_01.dat ../data/moomin_walk_02.dat
	Rscript --vanilla figure04.R ../data/moomin_walk_01.dat ../data/moomin_walk_02.dat
```

- When we run `make`, `figure04.pdf` is generated.

Makefile: wildcards


```
# makefile for figures for the paper
# "Fiji Earthquake and Moomin paintball"

all : output/figure01.pdf output/figure02.pdf output/figure03.pdf output/figure04.pdf

output/figure01.pdf : figure01.R config.R
          Rscript --vanilla figure01.R

output/figure02.pdf : figure02.R config.R
          Rscript --vanilla figure02.R

output/figure03.pdf : figure03.R ../data/paintball_data
          Rscript --vanilla figure03.R

output/figure04.pdf : figure04.R ../data/moomin_walk_01.dat ../data/moomin_walk_02.dat
          Rscript --vanilla figure04.R ../data/moomin_walk_01.dat ../data/moomin_walk_02.dat
```

- This list of dependencies can be simplified if we use a wildcard.

Makefile: wildcards


```
# makefile for figures for the paper
# "Fiji Earthquake and Moomin paintball"

all : output/figure01.pdf output/figure02.pdf output/figure03.pdf output/figure04.pdf

output/figure01.pdf : figure01.R config.R
          Rscript --vanilla figure01.R

output/figure02.pdf : figure02.R config.R
          Rscript --vanilla figure02.R

output/figure03.pdf : figure03.R ../data/paintball_data
          Rscript --vanilla figure03.R

output/figure04.pdf : figure04.R ../data/moomin_walk_*.dat
          Rscript --vanilla figure04.R ../data/moomin_walk_01.dat ../data/moomin_walk_02.dat
```

- This list of dependencies can be simplified if we use a wildcard.

Makefile: wildcards


```
# makefile for figures for the paper
# "Fiji Earthquake and Moomin paintball"

all : output/figure01.pdf output/figure02.pdf output/figure03.pdf output/figure04.pdf

output/figure01.pdf : figure01.R config.R
          Rscript --vanilla figure01.R

output/figure02.pdf : figure02.R config.R
          Rscript --vanilla figure02.R

output/figure03.pdf : figure03.R ../data/paintball_data
          Rscript --vanilla figure03.R

output/figure04.pdf : figure04.R ../data/moomin_walk_*.dat
          Rscript --vanilla figure04.R ../data/moomin_walk_01.dat ../data/moomin_walk_02.dat
```

- We have to specify those prerequisites in the action.
- $^ is a special variable containing the prerequisites.

Makefile: wildcards


```
# makefile for figures for the paper
# "Fiji Earthquake and Moomin paintball"

all : output/figure01.pdf output/figure02.pdf output/figure03.pdf output/figure04.pdf

output/figure01.pdf : figure01.R config.R
          Rscript --vanilla figure01.R

output/figure02.pdf : figure02.R config.R
          Rscript --vanilla figure02.R

output/figure03.pdf : figure03.R ../data/paintball_data
          Rscript --vanilla figure03.R

output/figure04.pdf : figure04.R ../data/moomin_walk_*.dat
          Rscript --vanilla figure04.R $^
```

- We have to specify those prerequisites in the action.
- $^ is a special variable containing the prerequisites.

Makefile: wildcards


```
# makefile for figures for the paper
# "Fiji Earthquake and Moomin paintball"

all : output/figure01.pdf output/figure02.pdf output/figure03.pdf output/figure04.pdf

output/figure01.pdf : figure01.R config.R
          Rscript --vanilla figure01.R

output/figure02.pdf : figure02.R config.R
          Rscript --vanilla figure02.R

output/figure03.pdf : figure03.R ../data/paintball_data
          Rscript --vanilla figure03.R

output/figure04.pdf : figure04.R ../data/moomin_walk_*.dat
          Rscript --vanilla $^
```

- But `$^` also contains `figure04.R`.
- We have to remove `figure04.R` from the action.

Makefile: wildcards


```
# makefile for figures for the paper
# "Fiji Earthquake and Moomin paintball"

all : output/figure01.pdf output/figure02.pdf output/figure03.pdf output/figure04.pdf

output/figure01.pdf : figure01.R config.R
          Rscript --vanilla figure01.R

output/figure02.pdf : figure02.R config.R
          Rscript --vanilla figure02.R

output/figure03.pdf : figure03.R ../data/paintball_data
          Rscript --vanilla figure03.R

output/figure04.pdf : ../data/moomin_walk_*.dat
          Rscript --vanilla figure04.R $^

output/figure04.pdf : figure04.R
```

- We can also separate the prerequisites into two rules.
- When several rules describe a target, the prerequisites are the union of all prerequisites.

Makefile: wildcards


```
# makefile for figures for the paper
# "Fiji Earthquake and Moomin paintball"

all : output/figure01.pdf output/figure02.pdf output/figure03.pdf output/figure04.pdf

output/figure01.pdf : figure01.R config.R
          Rscript --vanilla figure01.R

output/figure02.pdf : figure02.R config.R
          Rscript --vanilla figure02.R

output/figure03.pdf : figure03.R ../data/paintball_data
          Rscript --vanilla figure03.R

output/figure04.pdf : ../data/moomin_walk_*.dat
          Rscript --vanilla figure04.R $^

output/figure04.pdf : figure04.R
```

- This **should** work but somehow I didn't manage to run it correctly...

Makefile: wildcards


```
# makefile for figures for the paper
# "Fiji Earthquake and Moomin paintball"

all : output/figure01.pdf output/figure02.pdf output/figure03.pdf output/figure04.pdf

output/figure01.pdf : figure01.R config.R
          Rscript --vanilla figure01.R

output/figure02.pdf : figure02.R config.R
          Rscript --vanilla figure02.R

output/figure03.pdf : figure03.R ../data/paintball_data
          Rscript --vanilla figure03.R

output/figure04.pdf : ../data/moomin_walk_*.dat
          Rscript --vanilla figure04.R $^

output/figure04.pdf : figure04.R
```

- Other special variables are $@ (rule's target), $< (rule's first prerequisite) and $? (rule's out-of-date prerequisites).

Makefile: wildcards


```
# makefile for figures for the paper
# "Fiji Earthquake and Moomin paintball"

all : output/figure01.pdf output/figure02.pdf output/figure03.pdf output/figure04.pdf

output/figure01.pdf : figure01.R config.R
          Rscript --vanilla figure01.R

output/figure02.pdf : figure02.R config.R
          Rscript --vanilla figure02.R

output/figure03.pdf : figure03.R ../data/paintball_data
          Rscript --vanilla figure03.R

output/figure04.pdf : figure04.R ../data/moomin_walk_*.dat
          Rscript --vanilla $^
```

- So this is our current makefile.

Makefile: wildcards


```
# makefile for figures for the paper
# "Fiji Earthquake and Moomin paintball"

all : output/figure01.pdf output/figure02.pdf output/figure03.pdf output/figure04.pdf

output/figure01.pdf : figure01.R config.R
          Rscript --vanilla figure01.R

output/figure02.pdf : figure02.R config.R
          Rscript --vanilla figure02.R

output/figure03.pdf : figure03.R ../data/paintball_data
          Rscript --vanilla figure03.R

output/figure04.pdf : figure04.R ../data/moomin_walk_*.dat
          Rscript --vanilla $^
```

- We could also use wildcards here.

Makefile: wildcards


```
# makefile for figures for the paper
# "Fiji Earthquake and Moomin paintball"

all : output/figure*.pdf

output/figure01.pdf : figure01.R config.R
          Rscript --vanilla figure01.R

output/figure02.pdf : figure02.R config.R
          Rscript --vanilla figure02.R

output/figure03.pdf : figure03.R ../data/paintball_data
          Rscript --vanilla figure03.R

output/figure04.pdf : figure04.R ../data/moomin_walk_*.dat
          Rscript --vanilla $^
```

- We could also use wildcards here.

Makefile: wildcards


```
# makefile for figures for the paper
# "Fiji Earthquake and Moomin paintball"

all : output/figure*.pdf

output/figure01.pdf : figure01.R config.R
	Rscript --vanilla figure01.R

output/figure02.pdf : figure02.R config.R
	Rscript --vanilla figure02.R

output/figure03.pdf : figure03.R ../data/paintball_data
	Rscript --vanilla figure03.R

output/figure04.pdf : figure04.R ../data/moomin_walk_*.dat
	Rscript --vanilla $^
```

- If we modify `config.R`, `make` updates the correct files.

Makefile: wildcards


```
# makefile for figures for the paper
# "Fiji Earthquake and Moomin paintball"

all : output/figure*.pdf

output/figure01.pdf : figure01.R config.R
	Rscript --vanilla figure01.R

output/figure02.pdf : figure02.R config.R
	Rscript --vanilla figure02.R

output/figure03.pdf : figure03.R ../data/paintball_data
	Rscript --vanilla figure03.R

output/figure04.pdf : figure04.R ../data/moomin_walk_*.dat
	Rscript --vanilla $^
```

- If we do `make clean`, `make` will not regenerate the figures afterwards.

Makefile: wildcards


```
# makefile for figures for the paper
# "Fiji Earthquake and Moomin paintball"

all : output/figure*.pdf

output/figure01.pdf : figure01.R config.R
          Rscript --vanilla figure01.R

output/figure02.pdf : figure02.R config.R
          Rscript --vanilla figure02.R

output/figure03.pdf : figure03.R ../data/paintball_data
          Rscript --vanilla figure03.R

output/figure04.pdf : figure04.R ../data/moomin_walk_*.dat
          Rscript --vanilla $^
```

- This is because `figure*.pdf` do not exist on the hard drive when `make` reads the makefile, so there is nothing to update.

Makefile: wildcards


```
# makefile for figures for the paper
# "Fiji Earthquake and Moomin paintball"

all : output/figure*.pdf

output/figure01.pdf : figure01.R config.R
	Rscript --vanilla figure01.R

output/figure02.pdf : figure02.R config.R
	Rscript --vanilla figure02.R

output/figure03.pdf : figure03.R ../data/paintball_data
	Rscript --vanilla figure03.R

output/figure04.pdf : figure04.R ../data/moomin_walk_*.dat
	Rscript --vanilla $^
```

- `../data/moomin_walk_*.dat` works because those files are not built by `make` and always pre-exist the `make` run.

Makefile: wildcards


```
# makefile for figures for the paper
# "Fiji Earthquake and Moomin paintball"

all : output/figure01.pdf output/figure02.pdf output/figure03.pdf output/figure04.pdf

output/figure01.pdf : figure01.R config.R
          Rscript --vanilla figure01.R

output/figure02.pdf : figure02.R config.R
          Rscript --vanilla figure02.R

output/figure03.pdf : figure03.R ../data/paintball_data
          Rscript --vanilla figure03.R

output/figure04.pdf : figure04.R ../data/moomin_walk_*.dat
          Rscript --vanilla $^
```

- So here we have no choice and must keep the detailed dependencies.

Going global

- What about our manual tuning of `write.pdf`?
- We can modify each script so that they read an argument determining if the pdf should be produced or not.
- Let's modify the 3 first scripts (but not `figure04.R`) this way:

```
# --- file output ---
write.pdf = F # default
arguments = commandArgs(trailingOnly = T)
if (length(arguments) > 0 && arguments == "pdf") {
write.pdf = T
}
```

Going global

- Now we can run `make` and produce the pdf files or run the scripts from within RStudio without manually changing this flag.

- We did not apply this to `figure04.R` because we would need to insert the pdf argument between the script name and the data file names. It would be possible if we could separate rules but I didn't manage to do it.

```
# action wanted
Rscript --vanilla figure04.R pdf data01 data02 ...

# action in the rule
Rscript --vanilla $^

# We would need to put 'pdf' in the prerequisites, which we cannot do
```

Going global

- Now we can run `make` and produce the pdf files or run the scripts from within RStudio without manually changing this flag.

- We did not apply this to `figure04.R` because we would need to insert the pdf argument between the script name and the data file names. It would be possible if we could separate rules but I didn't manage to do it.

```
# what we could theoretically do

output/figure04.pdf : ../data/moomin_walk_*.dat
        Rscript --vanilla figure04.R pdf $^

output/figure04.pdf : figure04.R
```

Makefile: macros

- Another feature of make is the use of macros.
- Macros are variables defined within the makefile.

```
# makefile for figures for the paper
# "Fiji Earthquake and Moomin paintball"

# Rscript configuration
R_COMMAND = Rscript --vanilla

# rules
all : output/figure01.pdf output/figure02.pdf output/figure03.pdf ...

output/figure01.pdf : figure01.R
        $(R_COMMAND) figure01.R

output/figure02.pdf : figure02.R
        $(R_COMMAND) figure02.R

...
```

Makefile: macros

- This can also be used for directories.

```
# makefile for figures for the paper
# "Fiji Earthquake and Moomin paintball"

# Rscript configuration
R_COMMAND = Rscript --vanilla
# output directory
DIR = output

# rules
all : $(DIR)/figure01.pdf $(DIR)/figure02.pdf $(DIR)/figure03.pdf ...

$(DIR)/figure01.pdf : figure01.R
        $(R_COMMAND) figure01.R

$(DIR)/figure02.pdf : figure02.R
        $(R_COMMAND) figure02.R

...
```

Makefile: patterns

- The last feature we'll see today is patterns.
- Patterns can define rules to build similar files.

```
# makefile for figures for the paper
# "Fiji Earthquake and Moomin paintball"

# Rscript configuration
R_COMMAND = Rscript --vanilla
# output directory
DIR = output

# rules
all : $(DIR)/figure01.pdf $(DIR)/figure02.pdf $(DIR)/figure03.pdf ...

$(DIR)/figure%.pdf : figure%.R config.R
        $(R_COMMAND) $< pdf

$(DIR)/figure03.pdf : ../data/paintball_data

$(DIR)/figure04.pdf : ...
```

Makefile: patterns

- This works for `figure01.R`, `figure02.R` and `figure03.R` because the action is the same.

- If we were to include the data file names into the script for `figure04.R`, our whole makefile could be:

```
# rules
all : $(DIR)/figure01.pdf $(DIR)/figure02.pdf $(DIR)/figure03.pdf ...

$(DIR)/figure%.pdf : figure%.R config.R
        $(R_COMMAND) $< pdf

$(DIR)/figure03.pdf : ../data/paintball_data

$(DIR)/figure04.pdf : ../data/moomin_walk_*.dat
```

- All the actions are described by a single rule. The others just describe the specific dependencies for update.

In brief

- `make` is an efficient tool to automate figure building.
- `make` is an old program and its syntax can be quite cumbersome.
- Complex makefiles can be hard to debug, but simple ones are often enough for figure building.

In brief

- `make` is an efficient tool to automate figure building.
- `make` is an old program and its syntax can be quite cumbersome.
- Complex makefiles can be hard to debug, but simple ones are often enough for figure building.

In brief

- `make` is an efficient tool to automate figure building.
- `make` is an old program and its syntax can be cumbersome sometimes.
- Complex makefiles can be hard to debug, but simple ones are often enough for figure building.

Further reading and references

Further reading and references

Software carpentry

- A reference with very clear lessons on many topics
  http://software-carpentry.org/
  http://software-carpentry.org/4_0/make/

GNU make

- http://gnu.org/software/make/
- For windows: http://gnuwin32.sourceforge.net/packages/make/

Beyond `make`: GnuWin, general Unix-like tools for Windows

- http://gnuwin32.sourceforge.net/
- Very useful: http://gnuwin32.sourceforge.net/packages/coreutils.htm